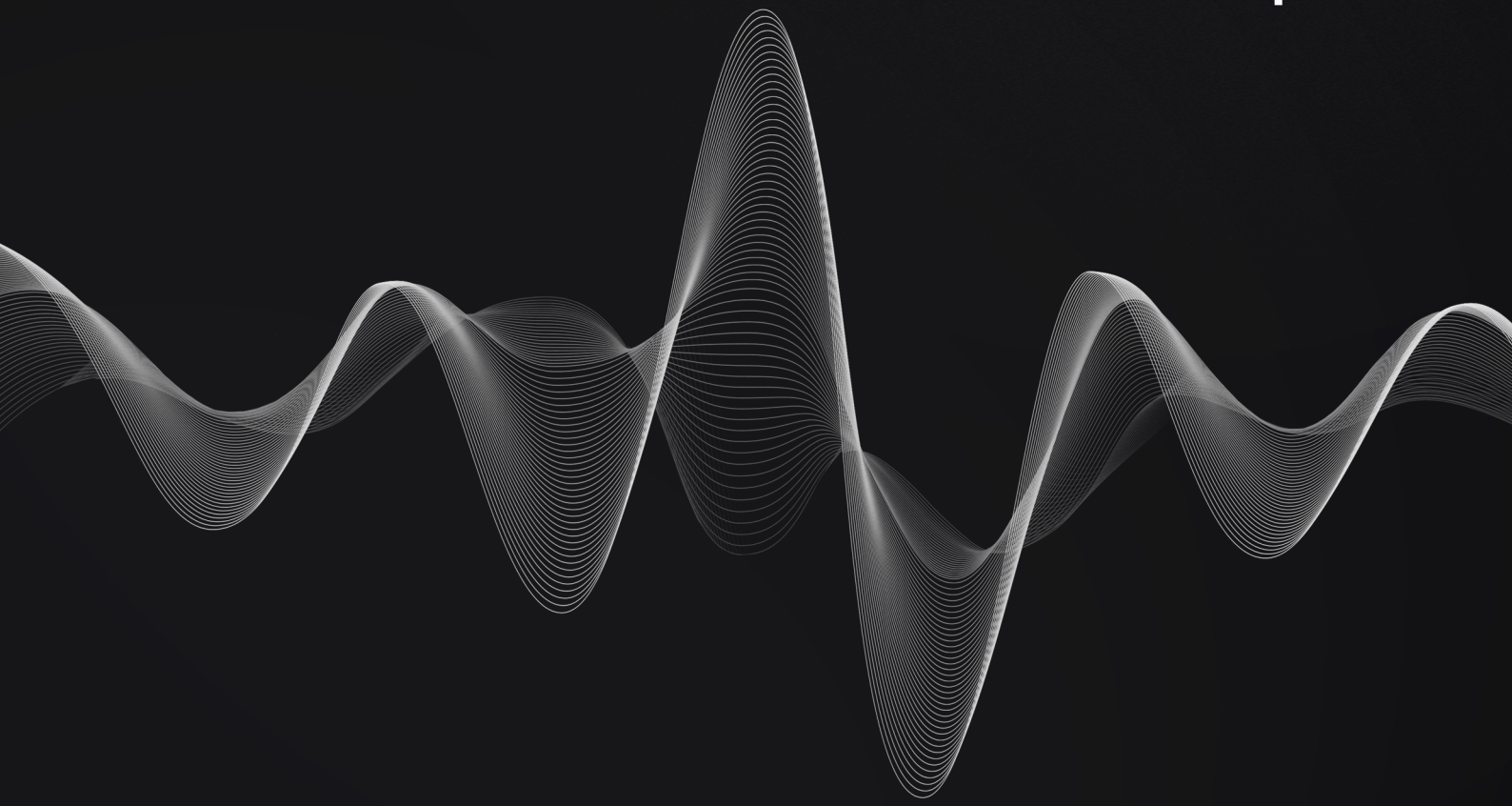
RESONANCE

CALC

Calculated Finance

Smart Contract Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

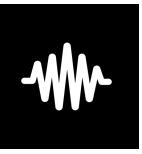
Document Control

PUBLIC **FINAL**(v2.0)

Audit_Report_CALC-APW_FINAL_20

Apr 3, 2024	v0.1	Michal Bazyli: Initial draft
Apr 4, 2024	v0.2	Michał Bazyli: Added findings
Apr 5, 2024	v1.0	Charles Dray: Approved
Apr 11, 2024	v1.1	Michal Bazyli: Reviewed findings
Apr 12, 2024	v2.0	Charles Dray: Published

Points of Contact	Fabrizio	Calculated Finance	Fabrizio@calculated.fi
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Michał Bazyli	Resonance	michal.bazyli@resonance.security
	Ilan Abitbol	Resonance	ilan.abitbol@resonance.security
	João Simões	Resonance	joao.simoes@resonance.security
	Michal Bajor	Resonance	michal.bajor@resonance.security



Copyright and Disclaimer

Contents

Executive Summary

Calculated Finance contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between April 1, 2024 and April 5, 2023. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

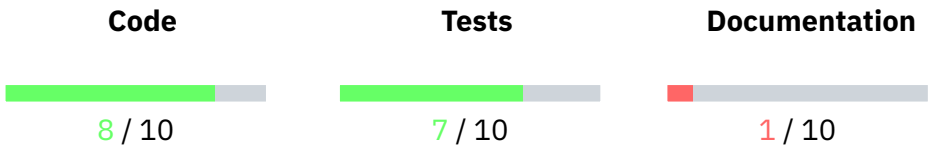
During the engagement, Resonance allocated 3 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 5 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Calculated Finance with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

System Overview

Calculated Finance provides machine-learning-powered risk averaging strategies written in CosmWasm which work on Kujira and Osmosis chains.

Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is good**.
- Unit tests are included. The tests cover both technical and functional requirements. Code coverage is undetermined. Overall, **tests coverage and quality is good**.
- The documentation is absent. Overall, **documentation coverage and quality is substandard**.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: `calculated-finance/calc/contracts/exchanges/astroport`
- Hash: 752452b661f27e69397adeab16f438d71f87181c

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Rust CosmWasm

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust CosmWasm audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Frontrunning attacks
- Unsafe third party integrations
- Denial of service
- Access control issues
- Inaccurate business logic implementations
- Incorrect gas usage

- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

Impact \ Likelihood	Very Likely	Likely	Unlikely
Strong	Critical	High	Medium
Medium	High	Medium	Low
Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
- **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Misleading Function Naming Regarding TWAP Simulation		Acknowledged
RES-02	Unnecessary admin Comparison In Migrate Function		Resolved
RES-03	Optional Parameters Should Not Be Optional		Acknowledged
RES-04	Redundant Logic In return_swapped_funds Due To Upstream Validation		Acknowledged
RES-05	Insufficient Validation Of Router Address On Contract Instantiation		Acknowledged
RES-06	"Migrate Only If Newer" Pattern Is Not Applied		Acknowledged
RES-07	Presence Of Non-Implemented Functionality In Contract Code		Acknowledged
RES-08	Presence Of Non-Implemented Query Functionality		Acknowledged
RES-09	Unnecessary Conversion In Exchange-Macros Library		Resolved
RES-10	Unnecessary Borrow in Astroport-Calc Library		Resolved
RES-11	Unnecessary Use Of to_string In Astroport-Calc		Resolved

Misleading Function Naming Regarding TWAP Simulation

Medium | RES-CALC-APW01 | Business Logic | Acknowledged

Code Section

- contracts/exchanges/astroport/src/handlers/get_twap_to_now.rs#L8

Description

The function named `get_twap_to_now_handler` in the `contracts/exchanges/astroport/src/handlers/get_twap_to_now.rs` code suggests that it calculates the Time-Weighted Average Price (TWAP) between two given denominations (`swap_denom` and `target_denom`). However, upon closer examination of the implementation details, it becomes evident that the function does not perform a TWAP calculation in the traditional sense. Instead, it simulates swap operations at fixed amounts to determine a current exchange rate or price using the Astroport protocol. This process is more akin to obtaining a spot price rather than calculating a TWAP, which typically involves averaging prices over a specified time period.

Recommendation

It is recommended to renaming the function to more accurately reflect its purpose and functionality. A name such as `simulate_swap_price` could provide clearer insight into the operation being performed, reducing the potential for confusion.

If a TWAP calculation is desired for certain use cases, consider implementing a separate function that averages price data over specified intervals. This would involve integrating or accessing historical price data and computing the average according to TWAP standards.

Status

The issue was acknowledged by Calculated Finance team. The development team stated "This is because we have a standardised msg interface that all swap contracts that our DCA contract interacts with must implement, obviously the restrictions of the underlying DEX affect what is and is not possible within the implementation.".

Unnecessary admin Comparison In Migrate Function

Low | **RES-CALC-APW02** | Business Logic | **Resolved**

Code Section

- contracts/exchanges/astroport/src/contract.rs#L38

Description

The `migrate` function within the specified contract performs an admin comparison check. This check compares the `admin` stored within the contract's configuration against the `admin` specified in the migration message (`msg.admin`). In the context of CosmWasm, this check is rendered superfluous due to the admin setting mechanism provided during the contract instantiation process.

CosmWasm treats contract migration as a first-class feature, allowing for an optional `admin` field to be set at the time of a contract's instantiation. This `admin` can be an external account or a governance contract, granted the exclusive right to initiate migrations. Should this field be left unset, the contract is deemed immutable, preventing any future migrations. The designated `admin` has the authority not only to execute migrations but also to alter the admin or even transition the contract to a fully immutable state.

Recommendation

It is recommended to remove the redundant admin comparison from the `migrate` function, streamlining the migration process under the assumption that the CosmWasm instantiation paradigm sufficiently secures migration permissions.

Status

The issue has been fixed in 8e1026d67478a4616ef8bb86547c0a0960870f2f.

Optional Parameters Should Not Be Optional

Low | **RES-CALC-APW03** | Data Validation | **Acknowledged**

Code Section

- contracts/exchanges/astroport/src/contract.rs#L71
- contracts/exchanges/astroport/src/contract.rs#L101
- contracts/exchanges/astroport/src/contract.rs#L116

Description

The `route` parameter for `ExecuteMsg::Swap`, `QueryMsg::GetTwapToNow`, and `QueryMsg::GetExpectedReceiveAmount` is initially treated as optional. This is inferred from the parameter's ability to assume a `None` value, suggesting that the functions could be called without specifying this parameter. However, a conflicting logic is observed immediately after this optional designation: the contract performs a check to ascertain if `route` is `None`, and if true, execution is halted with a `ContractError::Route` error.

This approach creates a logical inconsistency in how the contract handles the `route` parameter. By defining `route` as optional, the contract implies flexibility in its specification, yet the immediate enforcement check for a non-`None` value contradicts this flexibility. This discrepancy may lead to confusion among users and developers interacting with the contract, as it sets an expectation of optionality that is not supported by the contract's execution logic.

Recommendation

If the `route` parameter is indeed mandatory for the successful execution of `ExecuteMsg::Swap`, `QueryMsg::GetTwapToNow`, and `QueryMsg::GetExpectedReceiveAmount` it should not be treated as optional.

Status

The issue was acknowledged by Calculated Finance team. The development team stated "This is because we have a standardised msg interface that all swap contracts that our DCA contract interacts with must implement, obviously the restrictions of the underlying DEX affect what is and is not possible within the implementation.".

Redundant Logic In return_swapped_funds Due To Upstream Validation

Low | **RES-CALC-APW04** | Business Logic | **Acknowledged**

Code Section

- contracts/exchanges/astroport/src/contract.rs#L314

Description

The `return_swapped_funds` function in the `calc/contracts/exchanges/astroport/src/handlers/swap.rs` contains logic intended to ensure that the swap operation adheres to the minimum receive amount specified by the user. Specifically, it compares the post-swap balance of the target denomination with the pre-swap balance and the specified minimum receive amount. If the actual received amount is less than the minimum specified, the function returns an error.

However, this validation is redundant given the upstream validation performed by the Astroport router contract in `https://github.com/astroport-fi/astroport-core/blob/2f639c68e71a7569abe748fa3a787794357c2c1a/contracts/router/src/contract.rs#L135`. The router's execution logic, as specified in the provided Astroport core GitHub repository, already includes a check to ensure that the swap operation does not proceed if the swap amount is less than the specified minimum receive amount. If this condition is not met, the Astroport contract will revert the transaction with an `AssertionMinimumReceive` error, effectively rendering the post-swap validation in the `return_swapped_funds` function unnecessary.

Recommendation

It is recommended to remove the post-swap validation logic in the `return_swapped_funds` function that checks if the return amount is less than the minimum receive amount. Since the Astroport router already guarantees this condition through its execution logic, removing the redundant check will optimize gas costs and reduce the contract's complexity without compromising security or functionality.

Moreover altering the swap operation logic by specifying the `info.sender` as the recipient of the swap operation (i.e., changing `to: None` to `to: Some(info.sender)`), could reduce the code by obviating the need for the `return_swapped_funds` function. This modification ensures that swapped funds are directly transferred to the function caller.

Status

The issue was acknowledged by Calculated Finance team. The development team stated "We will keep this as it provides an extra layer of protection to the users and ensures the failure logic is consistent regardless of any changes to the underlying DEX implementation.".

Insufficient Validation Of Router Address On Contract Instantiation

Low | **RES-CALC-APW05** | Data Validation | **Acknowledged**

Code Section

- contracts/exchanges/astroport/src/contract.rs#L24

Description

The `instantiate` function in the `contracts/exchanges/astroport/src/contract.rs` code is responsible for initializing a smart contract with essential configuration details, including an `admin` address and a `router_address`. While the function does validate the format of these addresses using `deps.api.addr_validate`, it lacks thorough validation of the `router_address` to ensure its correctness and legitimacy as a router within the context it's supposed to operate in.

Recommendation

It is recommended to extend the validation logic to include checks beyond the address format. This could involve querying the blockchain to verify that the provided `router_address` corresponds to a contract that implements expected interfaces or functions. For instance, a minimal interaction with the `router_address` to call a known method (e.g., a version identifier or a method signature check) could ascertain its legitimacy as a router.

Status

The issue was acknowledged by Calculated Finance team. The development team stated "We will ignore this as even on chain verification of the router contract would not totally ensure that the correct address was passed in, and the swapper will fail pretty immediately if an incorrect router address is passed in, so we don't see it as a security risk.".

"Migrate Only If Newer" Pattern Is Not Applied

Info | **RES-CALC-APW06** | Business Logic | **Acknowledged**

Code Section

- contracts/exchanges/astroport/src/contract.rs#L38

Description

The contract migration function under examination does not implement the "Migrate When Newer" pattern, which is a recommended best practice within the CosmWasm ecosystem for contract upgrades.

The essential logic dictates that a migration should only proceed if the version of the contract being migrated to is newer than the currently deployed version. This check ensures that migrations are performed in a controlled manner, preventing potential downgrades or re-applications of the same version, which could lead to unintended state changes or data loss.

Recommendation

It is recommended to implement the "Migrate When Newer" pattern as part of the contract's migration logic. The migration should only proceed if the incoming version is newer than the current one, based on semantic versioning rules.

Status

The issue was acknowledged by Calculated Finance team. The development team stated "We will ignore this as a migrate if newer pattern adds complexity that we're not convinced is necessary, and we don't see it as a security risk.".

Presence Of Non-Implemented Functionality In Contract Code

Info	RES-CALC-APW07	Code Quality	Acknowledged

Code Section

- contracts/exchanges/astroport/src/contract.rs#L66-L96

Description

The smart contract code contains placeholders for `SubmitOrder`, `RetractOrder`, and `WithdrawOrder` functionalities within its message handling logic, as indicated by their explicit routing to a `not_implemented_handler()`. This approach suggests that the contract is designed to accommodate these functionalities, but they are currently not implemented. Including non-functional placeholders in the deployed contract can lead to confusion and misinterpretation of the contract's capabilities by users and developers. Moreover, it represents an unnecessary increase in the contract's complexity and size, potentially impacting its deployment and execution cost, albeit minimally.

Recommendation

It is recommended to remove non-implemented code, this will streamline the contract, reduce potential confusion, and ensure clarity regarding the contract's capabilities.

Status

The issue was acknowledged by Calculated Finance team. The development team stated "This is because we have a standardised msg interface that all swap contracts that our DCA contract interacts with must implement, obviously the restrictions of the underlying DEX affect what is and is not possible within the implementation.".

Presence Of Non-Implemented Query Functionality

Info	RES-CALC-APW08	Code Quality	Acknowledged

Code Section

- contracts/exchanges/astroport/src/contract.rs#L96-L97

Description

The contract interface includes definitions for `GetOrder` and `GetPairs` query functionalities that are currently not implemented, as evidenced by their handlers returning a `not_implemented_query()` response. This approach indicates that while the contract's interface suggests the availability of these query capabilities, they are, in practice, unimplemented. Including such non-functional stubs within the contract can create confusion regarding the contract's capabilities, misleading users and developers about its actual functionality. Furthermore, it adds unnecessary elements to the contract's interface, potentially complicating the codebase without offering practical utility.

Recommendation

It is recommended to remove non-implemented code, this will streamline the contract, reduce potential confusion, and ensure clarity regarding the contract's capabilities.

Status

The issue was acknowledged by Calculated Finance team. The development team stated "This is because we have a standardised msg interface that all swap contracts that our DCA contract interacts with must implement, obviously the restrictions of the underlying DEX affect what is and is not possible within the implementation.".

Unnecessary Conversion In Exchange-Macros Library

Info	RES-CALC-APW09	Code Quality	Resolved

Code Section

- packages/macros/src/lib.rs#L36

Description

An explicit call to `.into_iter()` is made on a variable `to_add` when passing it as a function argument where any `IntoIterator` is accepted.

Recommendation

It's suggested to remove the `.into_iter()` call since the parameter already accepts any `IntoIterator`, making this conversion unnecessary.

Status

The issue has been fixed in f726444cf3e9c8b357fe91f982e8b0487f048997.

Unnecessary Borrow in Astroport-Calc Library

Info	RES-CALC-APW10	Code Quality	Resolved

Code Section

- contracts/exchanges/astroport/src/handlers/swap.rs#L38

Description

The code unnecessarily borrows an expression `&route` that already implements the required traits for the function `from_json`.

Recommendation

It is suggested to change the borrowed expression to just `route`, removing the ampersand to avoid unnecessary borrowing.

Status

The issue has been fixed in f726444cf3e9c8b357fe91f982e8b0487f048997.

Unnecessary Use Of to_string In Astroport-Calc

Info	RES-CALC-APW11	Code Quality	Resolved

Code Section

- contracts/exchanges/astroport/src/handlers/swap.rs#L94

Description

There's an unnecessary use of `to_string` on `swap_cache.sender`, which could be replaced with a more efficient method.

Recommendation

Instead of converting `swap_cache.sender` to a string, it is suggested to use `swap_cache.sender.as_ref()`, which avoids the need for conversion.

Status

The issue has been fixed in f726444cf3e9c8b357fe91f982e8b0487f048997.

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.